Configuring IPvlan networking in DockerConfiguring IPvlan networking in Docker

https://4sysops.com/archives/configuring-ipvlan-networking-in-docker/

In my previous article, we discussed the MacVLAN network in Docker. Today, I will outline the differences between IPvlan and MacVLAN, explain the advantages of IPvlan, and show you how to create a VLAN with IPvlan.

Contents

IPvlan vs. MacVLAN

These are the differences between IPvlan and MacVLAN:

- The MacVLAN network allocates a unique MAC address to every container. Thus, a single network interface on a Docker host essentially advertises multiple MAC addresses. With an IPvlan network, all containers on a Docker host share a single MAC address.
- A MacVLAN network requires you to enable promiscuous mode on the parent interface of the Docker host, which is not required with IPvlan.

In addition to the above two distinctions, IPvlan can work in two modes: L2 mode and L3 mode.

L2 (or Layer 2) mode is the default mode for Ipvlan. It works just like MacVLAN (bridge) mode but without assigning a unique MAC address to each container. In L2 mode, the Docker host acts like a switch between the parent interface and a virtual NIC for each container. The entire communication is based only on MAC addresses. The containers in one IPvlan network (in L2 mode) can communicate with the containers in another IPvlan network. However, this generates many ARP broadcasts, which affects network performance.

In L3 (or Layer 3) mode, the Docker host works like a Layer 3 device to route the packets between the parent interface and the virtual NIC for each container. In this mode, containers are totally isolated from other networks, but the broadcasts are restricted to the Layer 2 subnet only; this improves network performance. The only downside is that you need to manually add a static route on your gateway router to let other network devices know how to reach your IPvlan network running in L3 mode.

Why an IPvlan network?

A MacVLAN network solves most use cases, but there are still some situations in which you might not want to or cannot work with MacVLAN. For instance, your old network switch doesn't support network interface advertising for many MAC addresses. Another point for IPvlan is that it works better in cloud environments, such as AWS or Azure.

Creating an IPvlan network

Take a look at the following diagram:

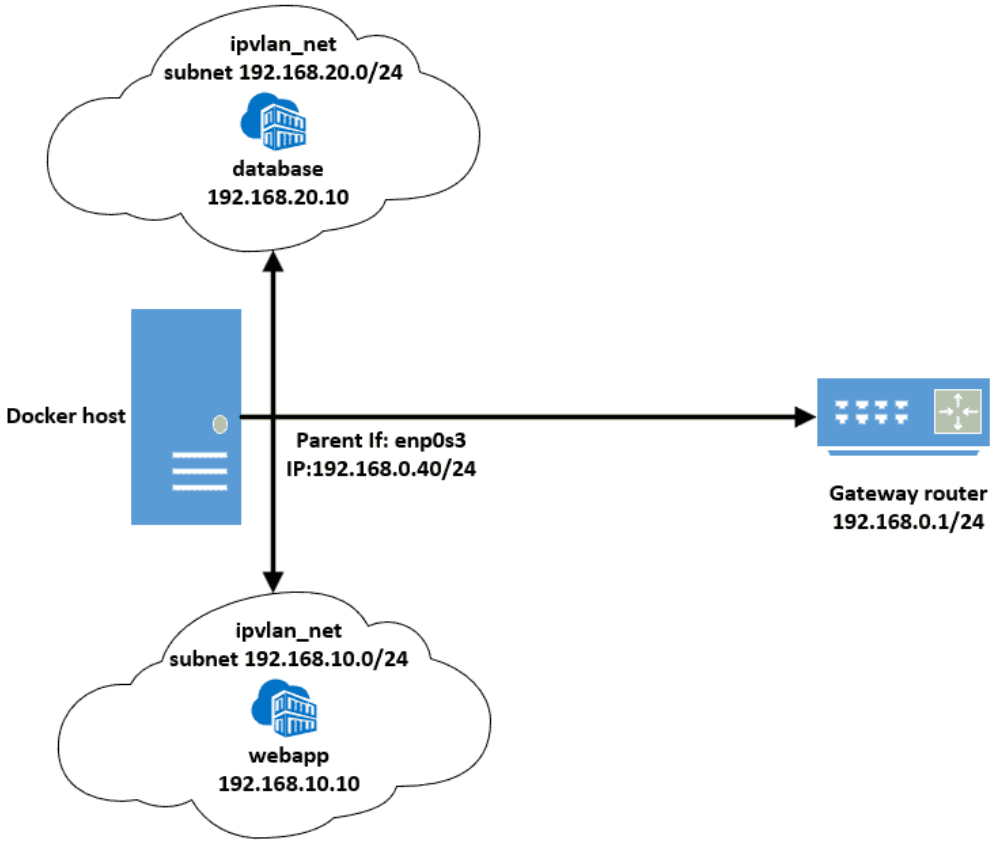


Diagram for demonstration of IPvlan configuration

In the diagram, you see one IPvlan network with two different subnets, one for each container. To create an IPvlan with this setup, use the *docker network create* command as shown below:

```
sudo docker network create \
--driver ipvlan \
--subnet 192.168.10.0/24 \
--subnet 192.168.20.0/24 \
--opt parent=enp0s3 \
--opt ipvlan_mode=l3 \
ipvlan_net
```

```
surender@srv2: ~
surender@srv2:~$
surender@srv2:~$ sudo docker network create \
--driver ipvlan \
--subnet 192.168.10.0/24 \
--subnet 192.168.20.0/24 \
--opt parent=enp0s3 \
--opt ipvlan_mode=l3 \
ipvlan_net
6adc69d40f054b11a2099b6cdc03c761b3cdef8c344b57f177bb6cc193a20a55
surender@srv2:~$
surender@srv2:~$
```

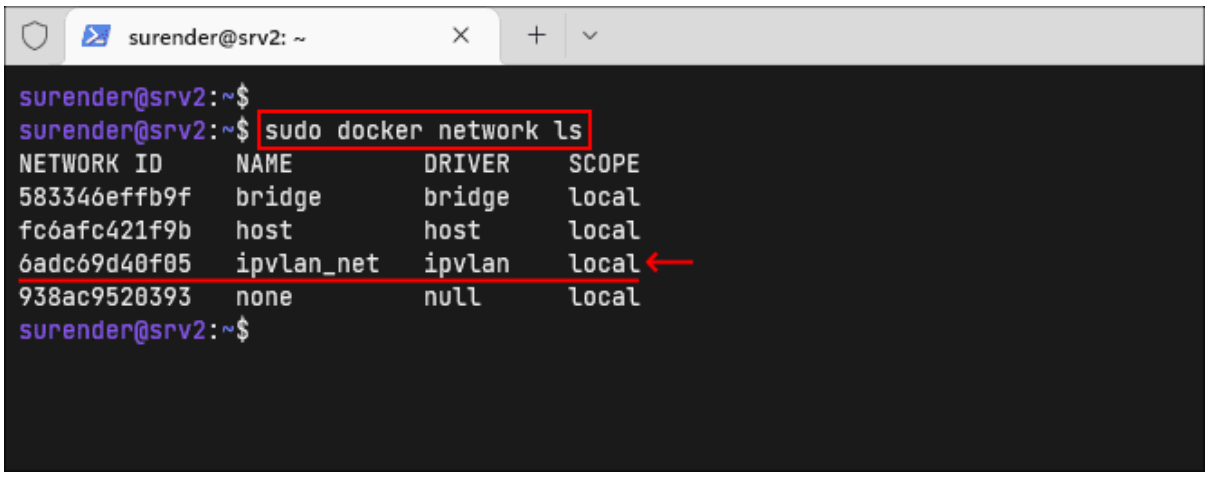
Creating an IPvlan network with L3 mode in Docker

Let's briefly discuss each option.

- *--driver* (or *-d*) is used to specify the IPvlan driver. In a previous post, we used the MacVLAN driver.
- *--subnet* specifies the subnet for the IPvlan network. Do you remember that the MacVLAN subnet must match the parent interface of the Docker host? The IPvlan network allows you to create a completely new virtual network right inside your Docker host. You can see that I used the *--subnet* option twice to create two brand new virtual networks. Other devices on my network don't see these subnets yet.
- *--opt* is used to specify additional options. Here, I used *parent=enp0s3* to specify the parent interface.
- The second most important option is to specify the IPvlan mode (*ipvlan_mode=l3)*. If you skip this option, your IPvlan network will essentially operate in Layer 2 (bridge) mode, since L2 mode is the default.
- At the end, we specify a name for our new IPvlan network (*ipvlan*net).

You just need to remember that you cannot create multiple IPvlan networks with one parent interface on the Docker host. One parent interface can serve only one IPvlan network. If you use one parent interface and define multiple subnets, as I did above, the containers of both subnets will be able to communicate with each other, by default, without any additional configuration. This allowed my *webapp* container with the subnet (192.168.10.0/24) to communicate with the *database* container, which is on a different the subnet (192.168.20.0/24). If you want to completely isolate subnets, you can use different parent interfaces for each subnet.

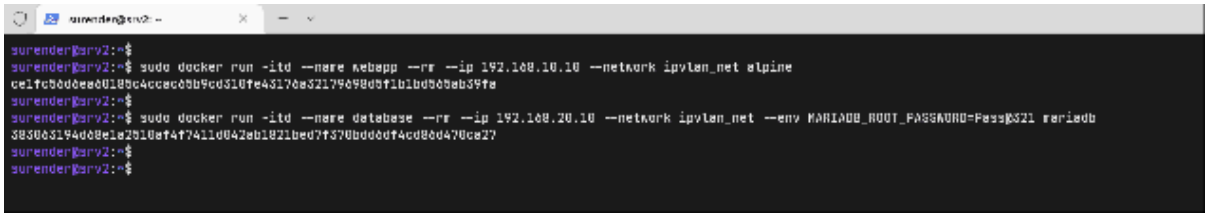
To view your newly created IPvlan network, use the *docker network ls* command.



Viewing Docker networks

Let's now quickly create two containers and attach them to our new IPvlan network.

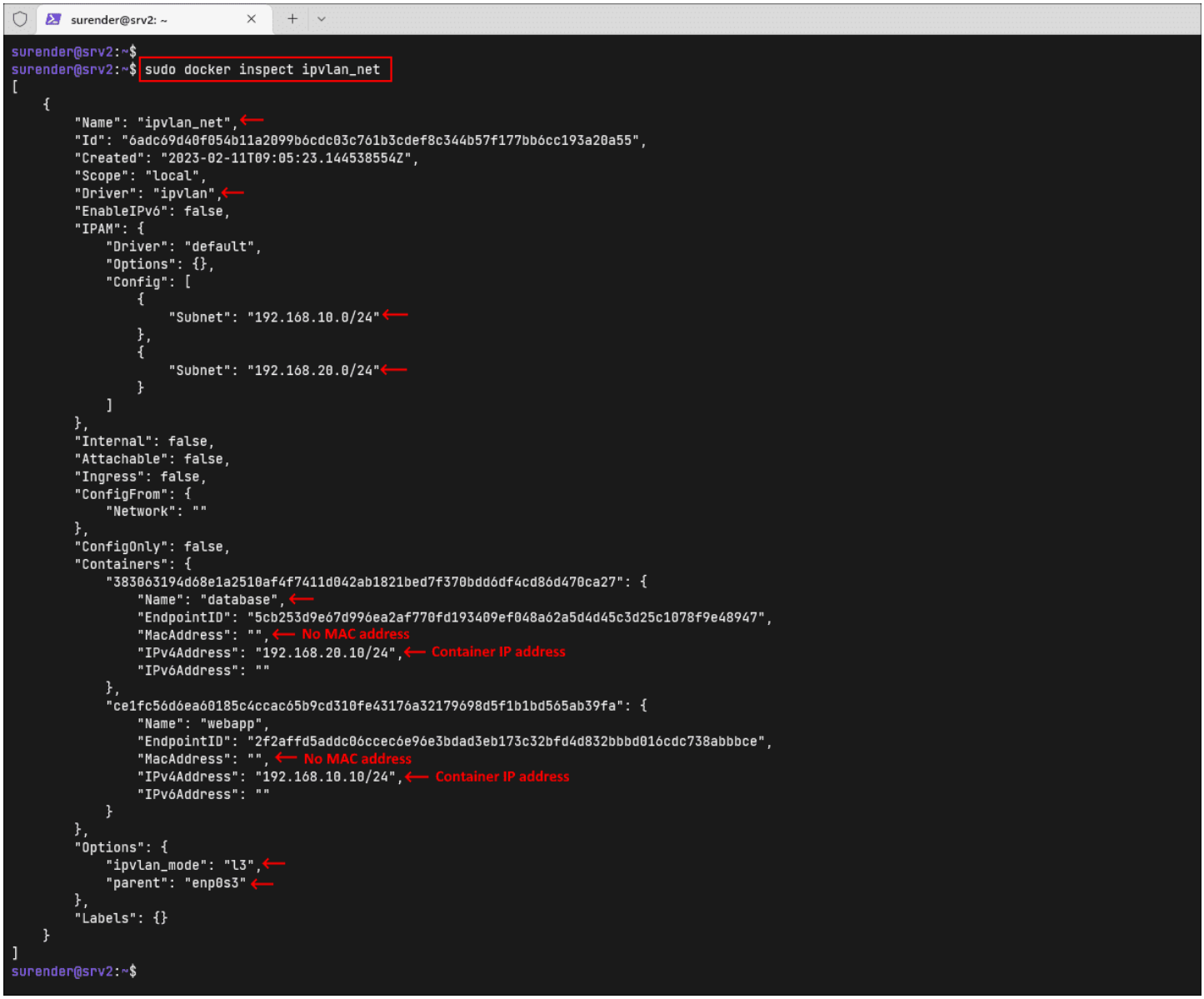
```
sudo docker run -itd --name webapp --rm --ip 192.168.10.10 --network ipvlan_net alpine
sudo docker run -itd --name database --rm --ip 192.168.20.10 --network ipvlan_net --env
MARIADB_ROOT_PASSWORD=Pass@321 mariadb
```



Creating new containers and attaching them to the IPvlan network

You can now inspect the Docker network using the *docker inspect ipvlan_net* command:

```
sudo docker inspect ipvlan_net
```

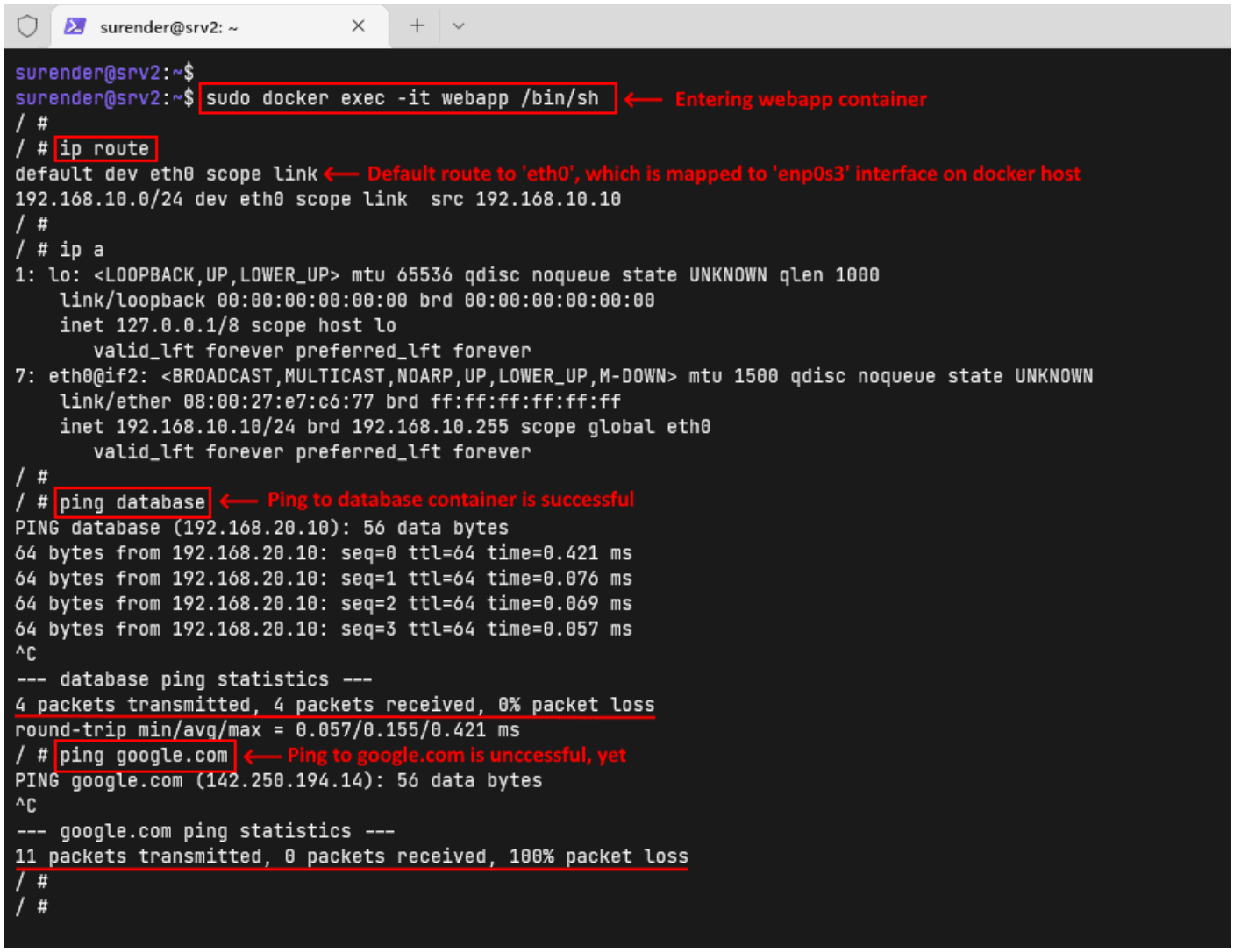


Inspecting the IPvlan network configuration

In the screenshot, you can see that both containers have IP addresses and there is no MAC address since our new Docker network is now operating in L3 mode. There are no MAC addresses and no ARP requests involved, which is really cool.

Verify connectivity

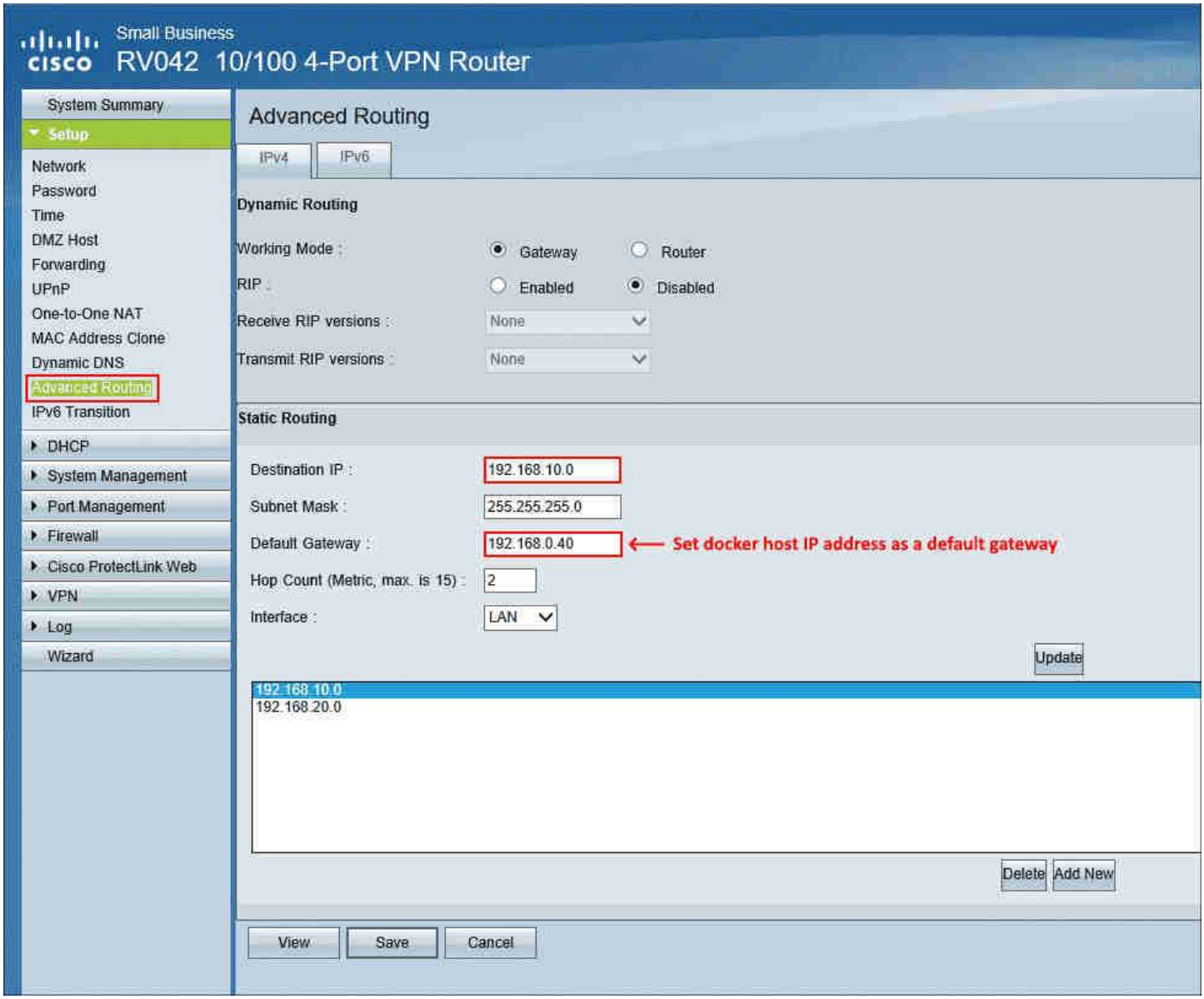
Let's now verify whether the containers attached to our new network can communicate. To do so, I will ping the *database* container and google.com from the *webapp* container.



Verify connectivity first attempt

As shown in the screenshot, the webapp container was able to ping the database container, but google.com was unreachable. So, why can't these containers access google.com even though there is a default route to the parent interface?

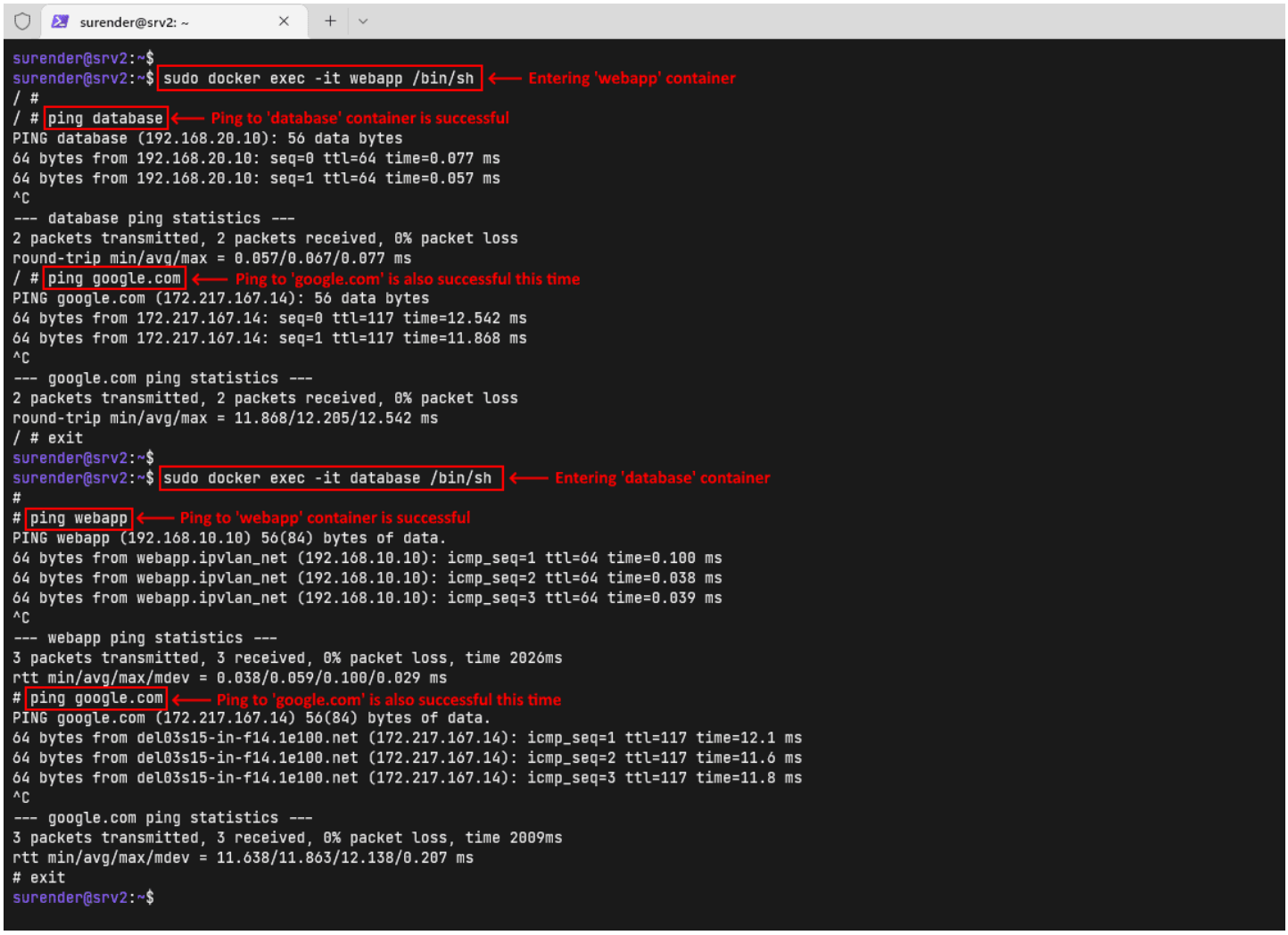
The reason is that our Docker host is on a different subnet (192.168.0.0/24) and they don't see the two new Docker subnets. To fix this, you need to log on to your gateway router and create a static route for each subnet.



Adding a static route for the Docker subnet to the gateway router

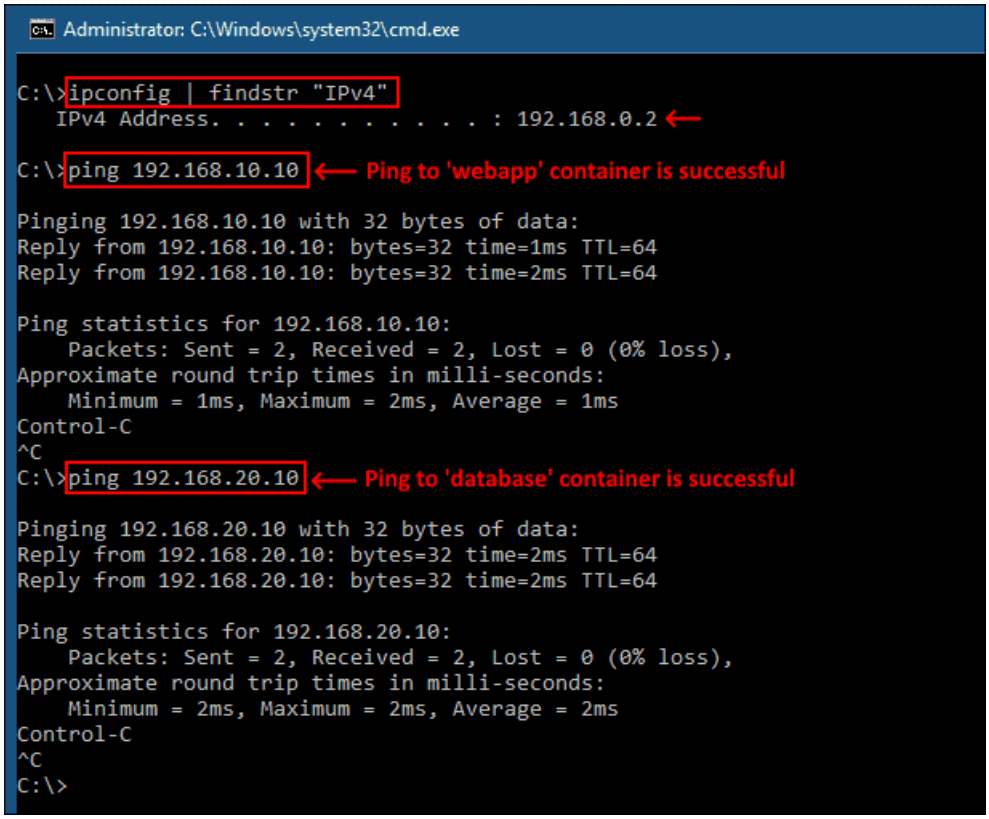
The important thing here is that you need to set the IP address of the Docker host as a default gateway or next-hop address, because your Docker host is now acting as a virtual router for the IPvlan network.

Once you do this, your Docker containers will be able to ping external hosts on the internet, as shown in the following screenshot.



Verify connectivity second attempt

Similarly, other devices on your network (*192.168.0.0/24*) will be able to communicate with both containers in the IPvlan network. In the following screenshot, you can see that a computer on my network successfully pinged both containers.



Verify connectivity from another host on the network

Conclusion

I hope you now understand the difference between MacVLAN and IPvlan in Docker and know how to configure your networks with both drivers. If you have questions, please leave a comment below.